VRIJE
UNIVERSITEIT
BRUSSEL

INFORMATION VISUALISATION

Project Report | Red List

Melba Raj S.M | *Melba.Raj.Sathyaraj.Mahila@vub.be* | 0574202
Akira Frédéric D Baes | *akirbaes@ulb.be* | 0549020

June 18, 2021

Professor: Prof. Beat Signer
Teaching Assistant: Maxim Van de Wynckel

Data management and analytics

Contents

Abstract

For the project of course *Information Visualisation*, The goal is to process and visualise a large data set using the theoretical knowledge we obtained from the lectures. We decided to visualise the red list data set. The data set contains the details of threatened organisms. We were able to produce three visualisations using various datasets on this theme using the knowledge we acquired from lectures. In order to evaluate our visualisations, We collected some feed backs from users working on different field.

1 Introduction

"Information visualisation is the process of representing data in a visual and meaningful way so that a user can better understand it" (*What is Information Visualization?*, n.d.) . In this project, the goal is to process and visualise a large dataset using the theoretical knowledge we obtained from the course lectures. We selected an interesting dataset, the Red List of Threatened Species. According to the BBC Wildlife Magazine, The Red List is the list of animals, plants and fungi that have been assessed for their risk of extinction (*What is the IUCN Red List? Here's everything you need to know*, n.d.). We collected this dataset from official IUCN (International Union for Conservation of Nature) website(*The IUCN Red List of Threatened Species*, n.d.). We selected this dataset because we found it very interesting. There have been many static visualisations of this data over the years but few interactive ones. The most noteworthy visualisation we found for the Red List is from the IUCN website itself, which functions more like a database search tool.(*The IUCN Red List of Threatened Species Search Page*, n.d.) We noticed that in the IUCN visualisation, we can mostly visualise the threatened location of animal/plant/fungi using their name. Since the data set was large data and also had multiple number of tables, we found this subject to be very interesting to visualise.

In this project, our target users are wild life conservation organizers and people knowledgeable about animal extinction. Although we didn't get any users related to wild life conservation, we tried to collect feed backs from users work/study related to Biological field.

We did three visualisations in this project. The first one is the visualization of statistical data of extinction per country. Secondly, the visualisation of the temporal data of the rate of extinction. Finally, the visualisation of worldwide extinction status based on animal habitat using spatial data.

2 Project Organization

In this session, we discuss about the task organized within our team during the project. We held weekly meetings on Microsoft Teams to discuss our ideas and work. For statistical data, preprocessing was an initial task that we had to done sequentially, but for the geodata, this was a continuously ongoing task to get the data workable enough. Most of the tables in this dataset were independent of each other. So after the data preprocessing, we worked in parallel. Initially we were a group of three people and worked together. After Interim presentation, one of our team member was not able to do further project due to personal reasons. We divided our workload together and there was a smooth communication and teamwork.

- Melba Raj S.M:
  - Data set collection
  - Preprocessing of Number of species tables listed in each IUCN Red List Category by country.

  - Visualization of total number of threatened organisms all over the world
  - Visualisation of the rate of extinction from 1996 to 2020
  - User feedback collection
  - Presentation slides preparation
  - Preparing project report

- Akira Frédéric D Baes:
  - Spatial Data sets collection
  - PDF to CSV conversion
  - Spatial Data simplification, aggregation
  - Visualisation of Spatial Data
  - Presentation slides preparation
  - Preparing project report

3 Frameworks

In this section, we discuss about the frameworks we used to implement this project.

- *Python 3* for implementing this project
- *Dash* is used as back end server
- *plotly 4.14.3* is used for plotting the visualisations
- *pycountry 20.7.3* library to create country codes and plot the corresponding country on world map.
- *pandas* to manipulate datasets
- *geopandas* library is used to control the spatial operations
- *Dbf5* to manipulate spatial complementary data in pandas without loading the geometric data (preprocessing)
- *maplotlib* and *geoplot* for previsualisation (preprocessing)

4 How to setup and run the visualisation

The visualisation was provided on canvas, and is also available on https://github.com/akirbaes/InfoVis2021
You will need conda and the libraries listed in group5_env.yml
Move a conda terminal to group5/plotly_geodata_app
(where this file, *group5_ environment.yml* and *main_ app.py* is located)
Create an environment with the necessary libraries with

```
conda create -f group5_env.yml
```

Switch to your environment

```
conda activate group5_env
```

This will launch the combined view:

```
python main_app.py
```

This will launch the geodata view only:

```
python geodata_app.py
```

This will launch a Dash server that ou can now open in your browser by visiting the given site:

```
Dash is running on http://127.0.0.1:8050/
```

5 Data set

In this section, we discuss about metadata of the dataset we used for visualisation.

5.1 Temporal data - table2_processed_main.csv

This table contains about 630 rows and 4 columns

Column name	Description
Year	The IUCN Red List over the last twenty years. However please note that there are many different reasons for these figures changing between different versions of The IUCN Red List.
Threatened_category	CR - Critically Endangered, EN - Endangered, VU - Vulnerable.
Classification_organism	In this column, the major taxonomic groups we considered for visualisation is included: Mammals, Birds, Amphibians, Fishes, Insects, Molluscs, Other invertebrates, Plants, Fungi & Protists. Also Total number of extinction rate of all these taxonomic groups are also included.
Number_of_Species	Threatened number of species for the respective taxonomic group.

Table 1: Metadata of temporal data of rate of extinction

5.2 Statistical data per country - Tab6_CountryCode.csv

This table contains about 1000 rows and 18 columns

column name	Description
Name	Name of the countries all over the world
EX	Extinct.
EW	Extinct in the Wild .
Subtotal(EX+EW)	Sum of Extinct and Extinct in the Wild .
CR(PE)	Critically Endangered(Possibly Extinct).
CR(PEW)	Critically Endangered(Possibly Extinct & Reintroduced).
Subtotal(EX+EW+ CR(PE)+ CR(PEW))	Sum of Extinct, Extinct in the Wild, Critically Endangered(Possibly Extinct), Critically Endangered(Possibly Extinct & Reintroduced).
CR	Critically Endangered.
EN	Endangered .
VU	Vulnerable.
Subtotal(threatened spp.)	sum of Critically Endangered, Endangered, Vulnerable .
LR/cd	Lower Risk/conservation dependent.
NT or LR/nt	Near Threatened (includes LR/nt - Lower Risk/near threatened).
LC or LR/lc	Least Concern (includes LR/lc - Lower Risk, least concern).
DD	Data Deficient.
Total	Total sum of all the threatened categories .
species	Kingdoms considered in this visualisation: Kingdom Animalia, Kingdom Plantae, Kingdom Fungi, Kingdom Chromista.
CODE	generated country codes for the respective country names.

Table 2: Metadata of temporal data of rate of extinction

5.3 Spatial database

From the https://www.iucnredlist.org/resources/spatial-data-download site we retrieveda total of 25,4 GB of habitat and extinction risk data, across 38 datasets (dispersed in 29 folders).

The datasets names: AMPHIBIANS, ANGELFISH, BLENNIES, BONEFISH_TARPONS, BUTTERFLYFISH, CLUPEIFORMES, GROUPERS, HAGFISH, MARINEFISH_PART1, MARINEFISH_PART2, MARINEFISH_PART3, PUFFERFISH, SEABREAMS_PORGIES_PICARELS, SHARKS_RAYS_CHIMAERAS, SURGEONFISHES_TANGS_UNICORNFISHES, SYNGNATHIFORM_FISHES, TUNAS_BILLFISHES, WRASSES_PARROTFISHES, FW_GROUP_PART1, FW_GROUP_PART2, FW_GROUP_PART3, FW_GROUP_PART4, FW_GROUP_PART5, FW_GROUP_PART6, MAMMALS, MAMMALS_FRESHWATER, CONUS, LOBSTERS, MANGROVES, REEF_FORMING_CORALS_PART1, REEF_FORMING_CORALS_PART2, REEF_FORMING_CORALS_PART3, SEAGRASSES, BIRCHES, MAGNOLIAS, MAPLES, TEAS, REPTILES.

Each datasets consists of:

- .dbf dBase database file containing most data except geometry (1 to 30 MB each)

- .shp a shape database containing geometrical information (several>1 GB)
- .cpg, .id_no_atx, .prj, .sbn, .sbx, .shp.xml, .shx, other structural files (<1 MB)
- "IUCN Red List Terms and Conditions of Use.pdf" which limits distribution of the dataset
- "METADATA_for_Digital_Distribution_Maps_of_The_IUCN_Red_List_of_Threatened_Species.pdf" which contains other constraints and geographical metadata

The .shp contains Multipolygons with holes in Unprojected Geographic Coordinate system, using Decimal degrees as unit (WGS_1984). RedListGIS@iucn.org for more informations.
The .dbf contains the following entries. The meaning of some entries is not provided.

Column name	Description
id_no	unique identifier
binomial	Binomial nomenclature of the compiled specie
presence	Number
origin	Number
seasonal	Number
compiler	Organism/person responsible for compiling the data
yrcompiled	The year the information was compiled
citation	The citation to use for crediting the data
subpop	Contains the name of a place (mosty empty)
source	External credit (mostly empty)
tax_comm	Common taxonomy, synonyms (mostly empty)
legend	Observation on the presence of animal (ranges from Extinct, Possibly Extinct, Presence Uncertain, etc. up to Extant (resident))
kingdom	Taxonomical kingdom
phylum	Taxonomical phylum
class	Taxonomical class
orde_	Taxonomical order
family	Taxonomical family
genus	Taxonomical genus
category	Extinction risk category two-letter classification
marine	Is the animal sea-based (true or false)
terrestial	Is the animal land-based (true or false)
freshwater	Is the animal river or lake-based (true or false)
SHAPE_Leng	Automatically generated Length data
SHAPE_Area	Automatically generated Area data

Table 3: Metadata of .dbf

There are 86 763 entries representing 50 035 binomial species (some species have several habitat which themselve are made of several Polygons).
Out of those informations, the binomial, taxon, extinction risk category and territory type data are the most useful to extract.
The .shp files are filled by Polygon and Multipylogon shapely objects. You can see more details about evey single file at 4.

6 Data Preprocessing

6.1 Converting statistical data from PDF

We used tabula, a python library, to turn PDF tables into CSV tables. However those tables were not fit to be used as dataframes in pandas, as

- It would not follow a column title content structure fit for databases due to various text between the tables
- Tables using sub-tables were not properly detected by tabula

Those CSV had to be individually re-processed using keyword matching to detect redundant lines and broken parts. Text was removed by detecting rows with only one column filled. Some smaller tables were fixed by hand.

6.2 Country codes

One of the table in the dataset contained name of countries which was hard to plot on a world map. we used *pycountry* library to create 3 letter country code for each countries to plot them on the world map. The respective code(file name: Tab6_withCode.py) is in Tab6_processed folder

6.3 Aggregating and simplifying heavy Geodata

The spatial database contains 86 763 entries for 50 035, totaling about 8 million polygons (and 18 million interior polygons, totaling 1700 million points). Just loading the whole database takes up to 15 minutes. See annex. 4 It is possible to use it for simple manipulations and generating static visualisations10.1, but to be able to use it in a dynamic visualisation, it needs to be heavily preprocessed, due to its weight and complexity.

Several methods were used to reduce the complexity of the data:

- Filter out irrelevant data
- Aggregate similar data
- Split the data in relevant categories

The first effort was to get rid of irrelevant information. First, some data were only represented as points. To have a consistent visualisation they were removed, however each of those were only 10MB at most. Second, the database has a lot of information that would not end up being used in the .dbf, like comments from the researchers, who discovered it. We tried cleaning this out, but this didn't reduce the complexity of the database, as the .dbf were not the most heavy part. As an example, in Maplotlib one file would take 1 minute to load and upwards to 15 minutes to plot.10.1 The first approach to reduce the shape complexity was to use Geopandas's "simplify" method. This reduces the number of points in each Polygon if they are closer than the given number. This greatly mangled the database, but brought a significant size decrease with simplify(1). 10.1

However to plot a whole database at once, we needed to do more. We then tried aggregating per-database (one out of 38), per-extinction level (8 10), using Geopandas's Dissolve. Using dissolve on the data was very slow, so I tried simplifying the data even more with simplify(3) before using Dissolve, which would greatly mangle the data. 10.1 Unfortunately dissolving some

of the databases would still take upwards to hours for a single dataset as the log fragment shows for only MAMMALS.shp. (roughly one hour 40 minutes). As I'd learn later, reason for that is that despite reducing the number of points, simplify doesn't reduce the number of polygons in a Multipolygon, and the complexity of Aggregation is as closely linked to the amount of shapes to combine as the number of points they are made of (since each polygon is compared to every other for intersection, leading to n^2 in a naïve implementation).

```
Open 18 SIMPLIFIED_DATA\MAMMALS.shp
13.3 s  Finished reading database
12.1 s  Pre-simplified(3)
6059.1s  Data dissolved
2.0  s  Simplified
      category                                           geometry
color
0           DD  MULTIPOLYGON (((-179.99900 -54.58140, -179.999...
1           LC  MULTIPOLYGON (((179.99900 -16.14624, 179.99900...
2           NT  MULTIPOLYGON (((139.28643 -34.26389, 139.30684...
3           VU  MULTIPOLYGON (((139.58461 -36.06409, 139.71840...
4           EN  MULTIPOLYGON (((36.70211 -5.58424, 38.86364 -5...
15.0 s  Output to  AGGREGATED_DATA/_agg_s3_MAMMALS.shp
Elements: 16 Shape: (8, 2)
```

Internally, Geopandas uses shapely's unary_union in dissolve. I have not been able to find the source code, which might have helped me understand the complexity of the issue.
Seeing how much damage to the Geometric data was done, I looked for alternative ways to aggregate geometric data. One way I explored was rasterisation: since the aggregated data would serve as a "preview" of some sort, turning the bigger databases into images would be enough to have a preview of it, and images would be faster to draw than multiple polygons, just like Google Maps Satellite view does. Using Maplotlib and plt.savefig, I generated images for all Extinction layers for all Databases, leading to a set of 225 transparent images ready to be loaded generated at 300dpi.10.110.1 Contrary to Shape aggregation, it also contains species concentration information for each area without much extra data required. The whole set would weight just 10 MB. Using a Choropleth Mapbox in Plotly I could easily load the images. However I could not get the image to align properly with Plotly's visualisation of the rest of my data. Inputting -180;-90;180;90 as the latitude/longitude data would lead to the image bugging out and not showing, and I also couldn't get the projections to match. After a lot discouraging of experimentation and asking for help I gave up on that direction.
Another optimisation I thought of was to not use Dissolve directly, but rather to use Dissolve on pairs of Entries, and then use those Aggregated pairs as base for the next round. This would in theory require log(n) call to Dissolve instead of n^2, and could maybe reduce the aggregation time depending on how unary_union was implemented. However in practice this took longer to do on some files. I still let my computer run for several days to aggregate 80% of the data before giving up.
After reflexion, I sumrised that the main issue was not the complexity of the polygons, but their sheer amount. The database contained a lot of data that would not end up being used, for example very small islands that would go beyond Plotly's default zooming limits, and very small holes in bigger polygons. I did a test run aggregating by taxonomical Order (instead of by database) and removing all polygons with an area smaller than 0.05.10.1 I also tried deleting the interior holes of the polygons, which were a significant part of the data but didn't actually improve the visualisation.10.1 Doing this before the aggregation step greatly sped up the

aggregation, which could be done on the whole database in mere hours (see aggregation logs). It also lead to the full database (now simplified) to be a lot smaller and also load faster. Hereunder loading a pre-simplified version of the database only takes a second, and aggregating it takes 7 minutes and half, while also including more mammals than the original file. 10.1

```
---- 02:54:56 ----
Open 8/10  SMALLER_DATABASE_CLEAN_2\MAMMALS.shp
0.6  s  read database, 5333  elements
433.6s  Aggregated down to 8
  category                                           geometry  counter
0       CR  MULTIPOLYGON (((26.88130 -30.66146, 27.64190 -...       123
1       DD  MULTIPOLYGON (((-179.99900 -54.58140, -179.999...       617
2       EN  MULTIPOLYGON (((22.11747 -13.80626, 22.26487 -...       428
3       EW  MULTIPOLYGON (((-10.91420 18.63752, -10.62866 ...         1
4       EX  MULTIPOLYGON (((-69.37592 -52.48828, -69.12791...         6
5       LC  MULTIPOLYGON (((-179.00000 -84.00000, -180.000...      3289
6       NT  MULTIPOLYGON (((-156.61766 56.99296, -156.6156...       359
7       VU  MULTIPOLYGON (((179.99900 -20.01051, 179.99900...       510
0.8  s  output to SMALLER_AGGREGATED_DATA/MAMMALS.shp
---- 03:02:12 ----
```

6.4 Organising geodata

As discussed earlier, the data were grouped in 38 datasets (29 folders). Some had expected names, like AMPHIBIANS, MAMMALS. Some of them had very specific names like SURGEONFISHES_TANGS_UNICORNFISHES that focuses on specific genuses, and some very broad like FW_GROUP meaning "freshwater group" which contains species from various classes/phylums/kingdoms. To the user, searching based on those names would be mostly meaningless.
Our first reflex was to learn about taxonomical classification of animals. We planned to do a hierarchical organisation, where the user could select a Kingdom, and have it show up, then select a Phylum, have it show up, etc. until Order which looked like the smallest smallest meaningful separation we could have. Here is how the database looks based on taxonomy.

kingdom	phylum	class	counter
ANIMALIA	ANNELIDA	CLITELLATA	10
	ARTHROPODA	BRANCHIOPODA	6
		INSECTA	6338
		MALACOSTRACA	3173
	CHORDATA	ACTINOPTERYGII	21740
		AMPHIBIA	14293
		AVES	4535
		CEPHALASPIDOMORPHI	59
		CHONDRICHTHYES	2457
		MAMMALIA	13394
		MYXINI	150
		REPTILIA	11454
		SARCOPTERYGII	9
	CNIDARIA	ANTHOZOA	827
		HYDROZOA	16

	MOLLUSCA	BIVALVIA	912
		GASTROPODA	3687
FUNGI	ASCOMYCOTA	LECANOROMYCETES	4
	BASIDIOMYCOTA	AGARICOMYCETES	1
PLANTAE	BRYOPHYTA	BRYOPSIDA	21
	CHAROPHYTA	CHAROPHYACEAE	14
	MARCHANTIOPHYTA	JUNGERMANNIOPSIDA	1
		MARCHANTIOPSIDA	4
	TRACHEOPHYTA	LILIOPSIDA	1600
		LYCOPODIOPSIDA	64
		MAGNOLIOPSIDA	1872
		POLYPODIOPSIDA	122

Unfortunately we ran into three issues:

- For each of those "view mode" we would have to generate one aggregated file. As said earlier, generated aggregated files was very taxing and long.
- Some of those classifications make sense for general people, but some are completely obscure. Some classes had too many orders to list, some had only one.
- Due to a bug into the code, after running the aggregation process for several days, I found out that some classes and phylums were missing from the result for some reason, meaning that I had to fix and re-do the whole process.

Due to those issues we gave up on using raw taxonomy for our visualisation organisation. The next idea was to base ourselves on the common species classification names used in Visualisation 1 and Visualisation 2, which would also lead to a more unified visualisation. Those names are not scientific appellations, but by looking up each group on Wikipedia and based on the taxonomical representation (to not ignore big groups) of our database we decided on the following grouping:

Column	Value	Grouping	Entries
class	AVES	BIRDS	4535
class	MAMMALIA	MAMMALS	13394
class	AMPHIBIA	AMPHIBIANS	14293
class	ACTINOPTERYGII	FISHES	21740
class	CHONDRICHTHYES	CARTILAGINOUS FISHES	2457
class	INSECTA	INSECTS	6338
class	MALACOSTRACA	CRUSTACEANS	3173
phylum	MOLLUSCA	MOLLUSCS	4599
kingdom	PLANTAE	PLANTS	3698
kingdom	FUNGI	FUNGI	5

The other species that didn't fit those classes were unfortunately omitted by the time being to simplify the processing. Most notables are class MYXINI (150, eels) PHYLUM CNIDARIA (843, jellyfish and corals) and ANNEDLIDA (10, ringed worms). A future version could include them.

7 Visualisations

In this section, we discuss about the visualisations we made for this project.

7.1 Visualisation 1 - Statistical data per country

In this visualisation, we visualised the statistical data of total number of the threatened animals all over the world.

7.1.1 What user can visualise?

Through this visualisation, the user can view the extinction status of each country. User can view the extinction of four major kingdoms: Kingdom Animalia, Kingdom Plantae, Kingdom Fungi and Kingdom Chromista. A dropdown box is provided to select the kingdom as per the user choice. Also on the map, User can hover over the country that he/she want to see the extinction status. The mouse hover data shows the total number of threatened organisms(according to the selected kingdom), the selected country name and country code. Also the user can click on the country to see the details of extinction. There are about 11 types of threatened categories and the respective details are shown on the side bar provided.

7.1.2 Application flow

When the application runs, A choropleth map, sidebar and dropdown is shown. choropleth map for Kingdom Animalia is shown by default. When user selects another Kingdom via dropdown, the respective kingdom data is filtered from the table and are shown with the provided colors. When the user click on a country, the extinction details of the respective country is extracted from the table and sent to the front end. It displays on the sidebar provided.

7.1.3 Visualisation design

We used choropleth map for this visualization. We selected choropleth map because it is a good choice to show the intensity of extinction rate. we used four different colours for four Kingdoms. For each kingdoms, we used four luminance of the respective colour. Since we use four luminances, they are easy to recognize distinctly. The darker the colour, the more the extinction rate will be. The colour codes used for respective kingdoms are shown in the *Figure 2.*

7.1.4 Implementation Details

At first, the tables were four different tables for each kingdom.At the preprocessing stage, we merged them all together into one table with added an extra column for specifying which kingdom, the corresponding row data belongs to(code file name: process_table_6_into_one.ipynb in folder Table6_processed). Also we added three letter country codes for each country using *pycountry* library. We used plotly function *go.Choropleth()* with customized visual features to plot the visualisation. While click on a country, we pass the click-event to the back end and the *@app.callback()* returns the corresponding output which is displayed in the sidebar. We implemented a dropdown button with options to select the Kingdoms. While selecting each option, the kingdom data corresponding to the option is filtered from the table.

```
graphToShow=[animalFig,plantFig,fungiFig,chromistFig]
labels = ["Animal","Plant","Fungi","Chromist"]
# is going to be visible
visible = np.array(labels)
traces = []
buttons = []
for value in labels:
```

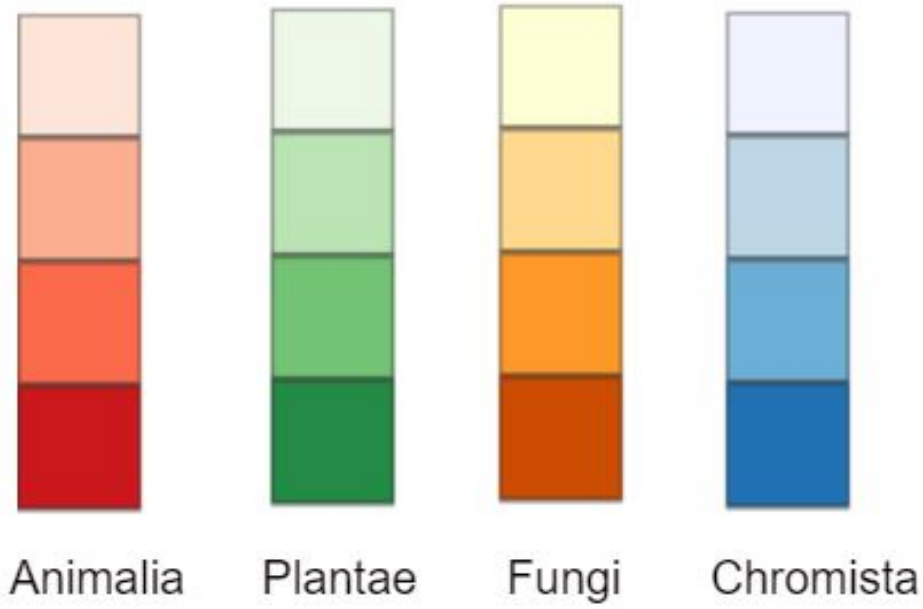


Figure 1: Color code used for choropleth map for each kingdom

```
    buttons.append(dict(label=value,
                        method='update',
                       args=[{"visible":list(visible==value)},
                              {"title":f"<b>{value}</b>"}
                        ]
                    ),
                )
updatemenus = [{"active":0,
                "buttons":buttons,
                "direction": 'up',
                }]
# Show figure
speciesFig = go.Figure(data=graphToShow,
                layout=dict(updatemenus=updatemenus))
```

7.1.5 Snapshots of the Visualisation 1-Statistical data per country

The snapshots of this visualisation are shown in Figure 2, 3, 4 and 5

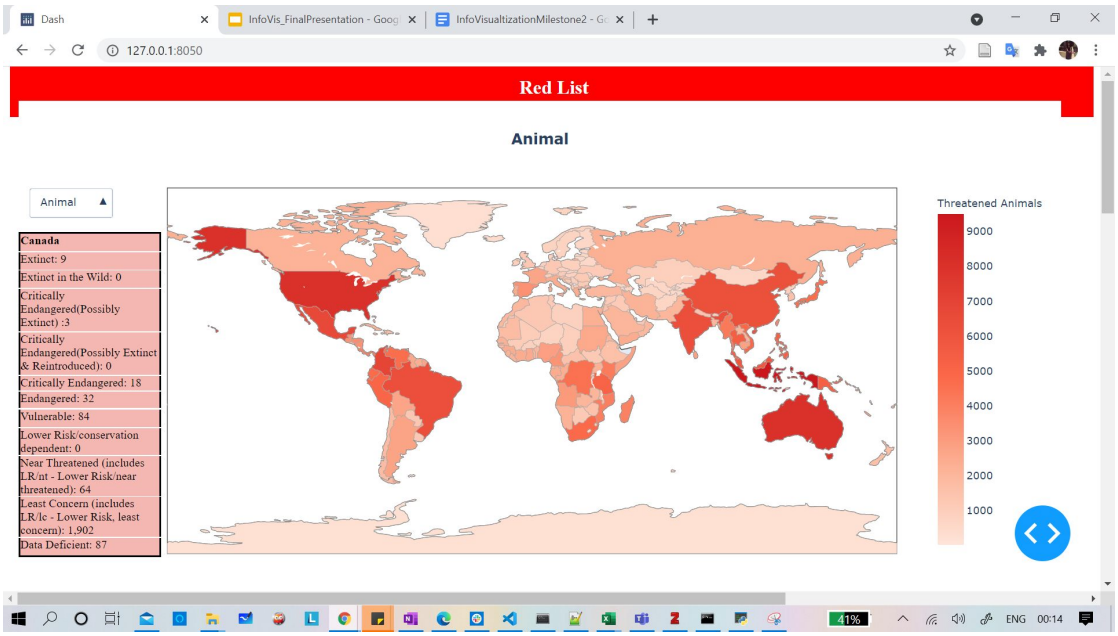


Figure 2: Kingdom Animalia with extinction details of Canada on side bar

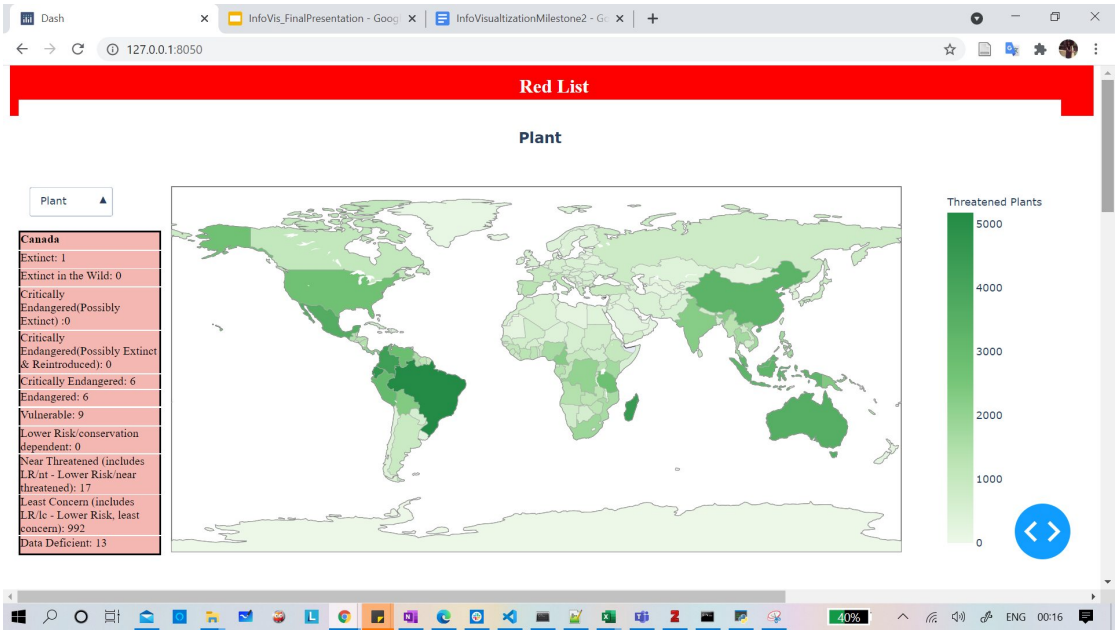


Figure 3: Kingdom Plantae with extinction details of Canada on side bar

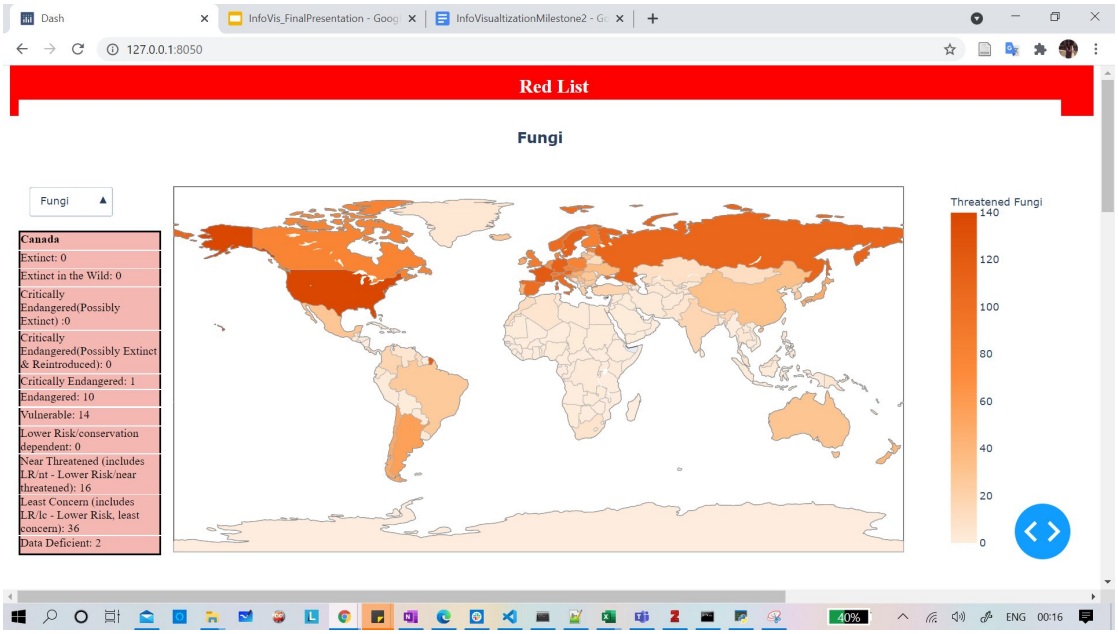


Figure 4: Kingdom Fungi with extinction details of Canada on side bar

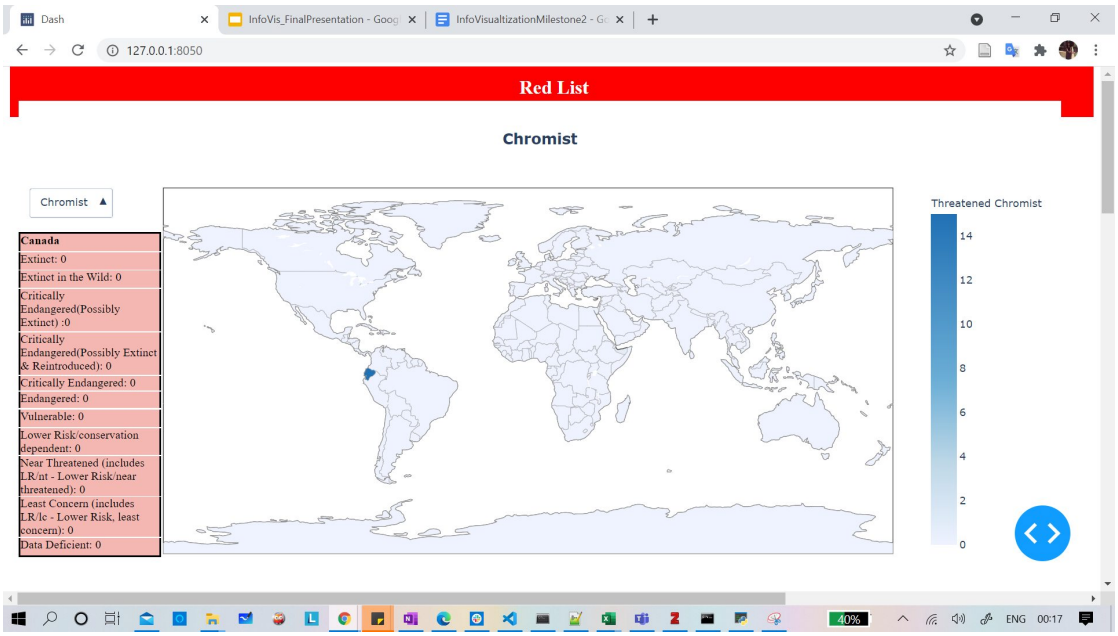


Figure 5: Kingdom Chromista with extinction details of Canada on side bar

7.2 Visualisation 2 - temporal data

In this visualisation, we visualised the temporal data of rate of extinction from 1996 to 2020.

7.2.1 What user can visualise?

Through this visualisation, the user can view the rate of extinction over the period from 1996 to 2020 of major taxonomic groups. User can view the variation of this rate of three threatened categories: Critically Endangered(CR), Endangered(EN), Vulnerable(VU). The radio button is provided to select the threatened category as per the user choice. Also on the bar plot, User can hover over the bars that he/she want to see the extinction rate of specific taxonomic group. The mouse hover data shows the total number of threatened of specific organism as per the year that varies in the time line,the taxonomic group and the corresponding year. the selected country name and country code. User can play the timeline button to visualise the variation of extinction rate from 1996 to 2020. If the user pause at specific year, he/she can statically visualise the extinction rate for that specific year. Also the user can click and drag on the individual bars to see the the variation of extinction rate. There are about 9 different taxonomic group that are plotted on bar plot. Also the total bar shows the total number of extinction rate as per the year.

7.2.2 Application flow

When the application runs, A bar plot and a series of three radio button is shown. Bar plot for threatened category- Critically Endangered(CR) and year 1996 is shown by default. When user plays the timeline, the bars in the plot vary according to the changing year in timeline. When user select another radio button, the data for the corresponding threatened category filtered from the table and the bar plot will be plotted.

7.2.3 Visualisation design

We used plotly animated bar charts for this visualization. We selected animated bar charts because it is a good choice to show the variations of extinction over a long period and will be very interesting to visualise. we used distinct colours for each taxonomic group so that each of them can be visualised without any ambiguity. The colour codes used for respective taxonomic groups are shown in the *Figure 6*.

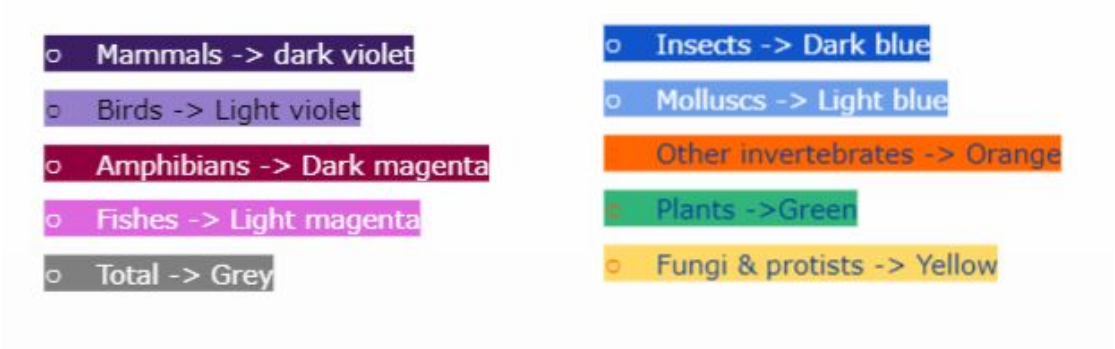


Figure 6: Color code used for Bar chart for each taxonomic group

7.2.4 Implementation Details

At first, the table was in PDF format.At the preprocessing stage, we converted them into CSV using *tabula*. Also we did some fixations by hand to make them into perfect CSV format. While converting to CSV using code, it contained 3 different tables for each threatened category. We merged them together into one table with adding an another column to mention which threatened category the row data belongs to. We used plotly function *px.bar()* with customized visual features to plot the visualisation. While play the timeline, the inbuilt feature of plotly barchart will show the variations of extinction rate from 1996 to 2020. If the user pause at specific year, he/she can statically visualise the extinction rate for that specific year. This is also the inbuilt feature of plotly animated bar charts. when we select a particular threatened category, the radio Items will pass the active information of the category we clicked and the *@app.callback()* returns the corresponding bar chart on the front end.

7.2.5 Snapshots of the Visualisation 1-Statistical data per country

The snapshots of this visualisation are shown in Figure 7, 8 and 9.

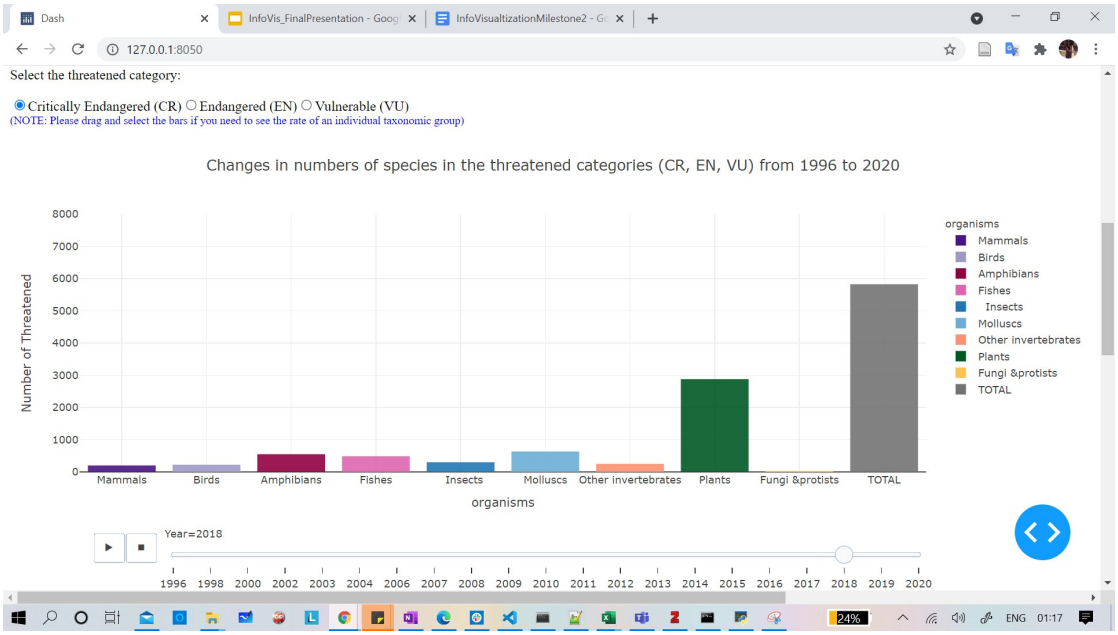


Figure 7: Extinction rate of Critically Endangered(CR) at 2018

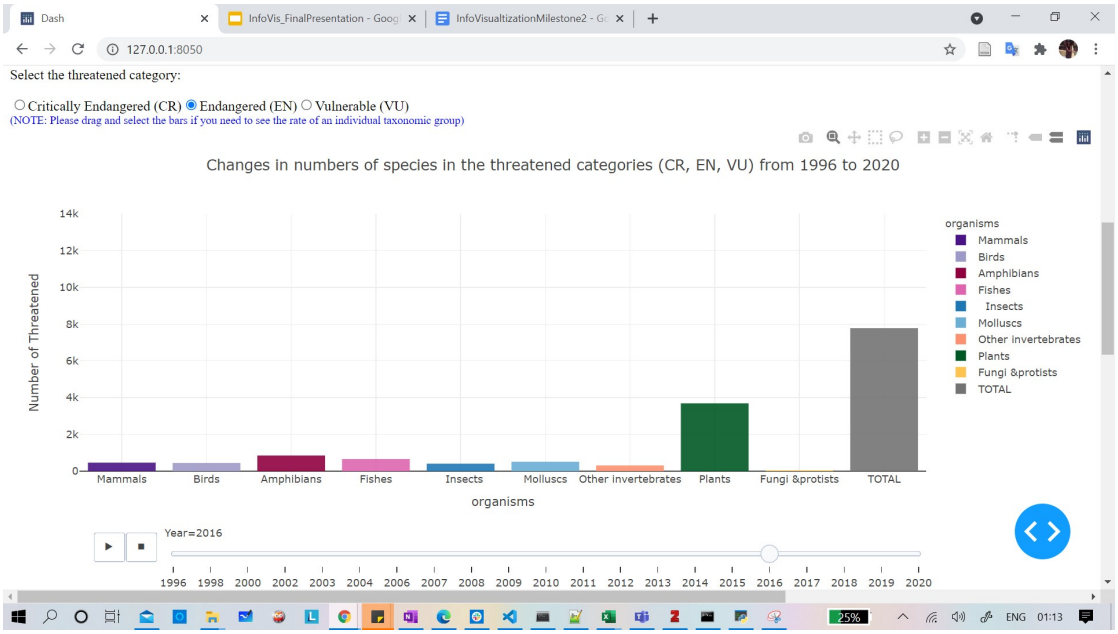


Figure 8: Extinction rate of Endangered(EN) at 2016

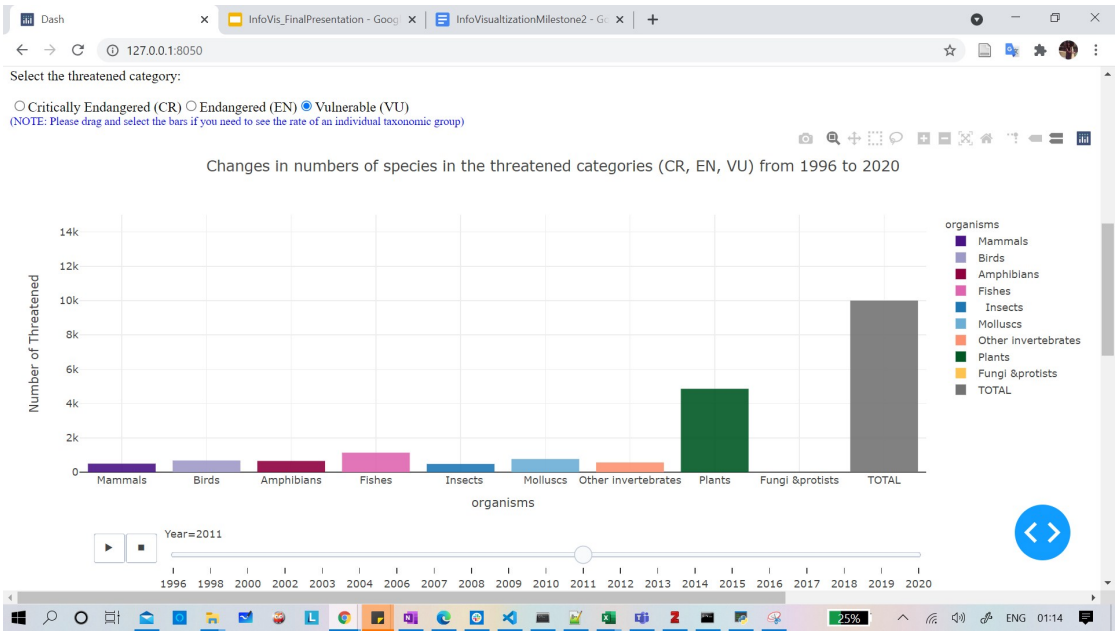


Figure 9: Extinction rate of Vulnerable(VU) at 2011

7.3 Visualisation 3 spatial data

7.3.1 User use-case

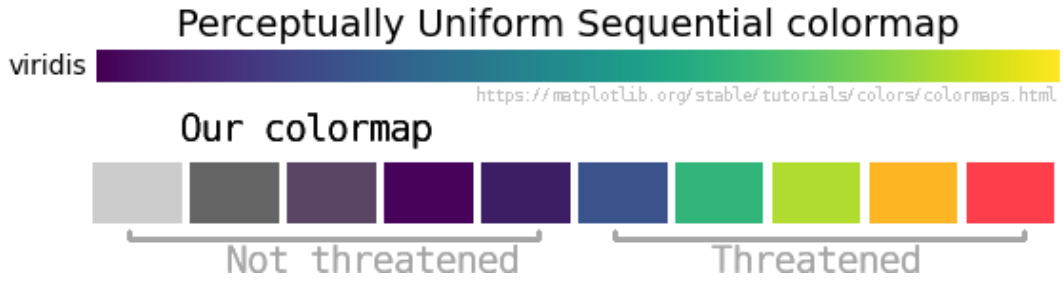
The third visualisation consists of 3 forms of user input and 2 sorts of data output used in 2 different modes.
The user can select a Database and several Extinction Level (color-coded) to search in. The user can move and zoom on the map.
In Aggregated mode, a preview of the selected database is shown. This might take long to load. A full list of animals present in the database is also output in a table.
In Selection mode, more precise information is given about the selected area in the center of the screen. The table will show a reduced count of the animals.

7.3.2 Application flow

The database first loads all the available Shapes data. This can take several minutes depending on the databases selected. The data is not dynamically loaded as it is required, because the application is meant to be a server that is initialised once and then left running, with users connecting to it. During testing we'd stick to smaller databases.
When clicking on a Database, its animals are sent to the table. This is relatively quick. At the same time, the aggregated form of the database is sent to a Plotly graph. This takes a while (about a minute or more for bigger databases).
When selecting an area, its coordinates are used to determine a slice of data to show. This is relatively quick, as the list of animals show up in the table. However sending the areas to plotly is, again, slow. Furthermore, moving quickly sends several coordinates to plotly which has to process the old request before moving on to the next. Plotly by default does not manage this well. A better threading/thread kill implementation would be required to fix this part, but since the bottleneck were on Plotly's renderer end I was not sure how to fix it. If I had more control over the data flow, maybe I could send shapes one by one to the renderer, while polling for changes in a separate thread.

7.3.3 Visualisation design

The color scheme is based on "viridis" of maplotlib, but modified to emphasize more important areas (extinctions in red) and deemphasize unimportant zones (animals not in danger in gray/dark). The two added colors (orange and red) are for really outstanding data, "Extinct in the wild" and "Extinct".



7.3.4 Implementation details

Geometrical shapes are sorted by area, then extinction level, to be able to see most shapes at the same time, and have the most important information on top. We have also noticed that the

closer to extinction an animal is, the smaller the habitat becomes.

```
def get_color(cat): return palette.index(cat)
geodata["color"]=geodata["category"].apply(get_color)
def get_area(shape): return shape.area
geodata["area"]=geodata["geometry"].apply(get_area)
geodata = geodata.sort_values(["color","area"],ascending=[True,False])
```

When selecting the habitat of an animal based on a point, we first explode the Multipolygon in individual polygons. This is to have the smallest, most precise habitat information only, because some multipolygons can span the entire planet. However the multipolygons are not always split evenly.

```
point=gpd.GeoDataFrame(geometry=gpd.GeoSeries(Point(point_coords)))
point.set_crs(epsg=4326,inplace=True)
intersection = alldata[alldata["origin"]==selected_database_name].explode().reset_index()
intersection = gpd.sjoin(geodata, point, op='intersects')
```

Selecting the data is done trough a callback of

```
Input("map_graph","relayoutData"))
```

from Plotly's graph. The reason is that while you can get a callback for clicking on a shape in Dash, you can't get one for clicking on an empty space, and the click doesn't give you specific coordinates if you click on a big polygon. I found an example of a hacky method to retrieve coordinates that was using a Javascript version of Plotly+Dash, but I couldn't translate it to Python. Dash also used to return coordinates when using the Selection tool's range data, but it doesn't as of more modern versions.

7.3.5 Snapshot of the Visualisation 3 animal habitat and danger level

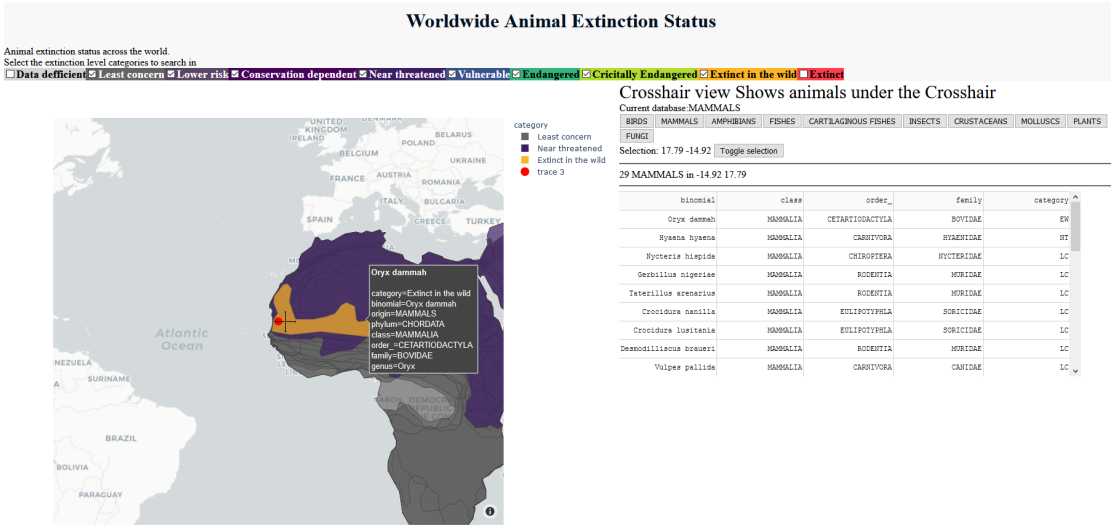


Figure 10: Snapshot of the spatial data visualisation

8 User Feed backs

In this section, we will discuss about the feed backs we collected from users.

8.1 How did we collect feedback?

We collected feedback from about 9 users for the first two visualisations we made. Our users are mostly between 20-29 age range. Since we did not host our visualisation, We conducted a Microsoft teams meeting with the user and showed our visualisations via sharing the screen, Then we collected feed backs via emailing them the feedback forms. We used UEQ questionnaire for the feed backs.

8.2 User responses and the changes we done

We got mixed feed backs. Some of the main feedback points we got are listed below:

- We didn't take into account "secure/Not secure" factor since we didn't host our visualisation
- Some of the users were not familiar with taxonomical hierarchy. They had a bit difficulty to understand.
- For the animated bar chart, the colours we initially used caused some ambiguity for users. So we changed them to distinct colours for each bar.
- One of the feedback said, if the user is not aware of how to use plotly, they will not think of drag and selecting the bars in bar plot for individual visualisation. So we added a notification just below the radio buttons to "drag and select the bars for individual visualisation".

8.3 User responses in graphs

8.3.1 Distribution of answers per item

The respective graph is shown in Figure 11. we got more positive feed backs on user friendly, attractive, organised, interesting and valuable. we got some "to be improved" feed backs on making the visualisations more easy and easily understandable.

8.3.2 Importance ratings

The respective graph is shown in Figure 12. We got about 5.64 rating on Attractiveness, 6.15 on Perspicuity, 6.44 on Efficiency, 6.22 on Dependability, 5.09 on Stimulation, 5.32 on Novelty. The rate is comparitively low for stimulation, which means growth to greater activity. We are not exact sure of the reason. But we think it's because some users had a bit dificulty to understand the visualisation.

9 Conclusion

To conclude, for this project of course *Information Visualisation*, we did three visualisations. We implemented our visualisation using the theoretical knowledge we acquired from lectures. We did visualisations on three different types of data: Statistical data, temporal data and spatial data.

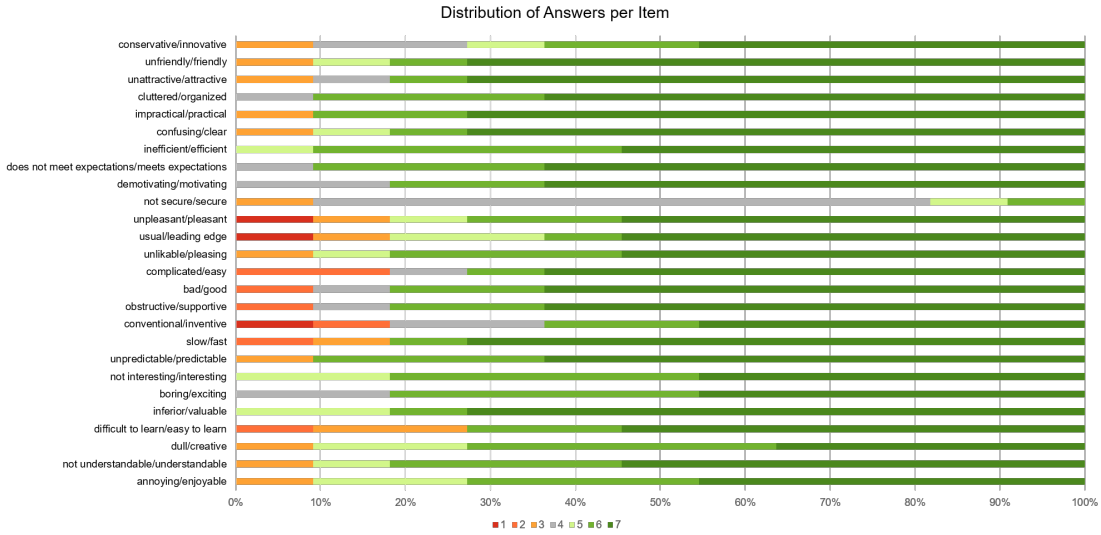


Figure 11: Distribution of answers per item

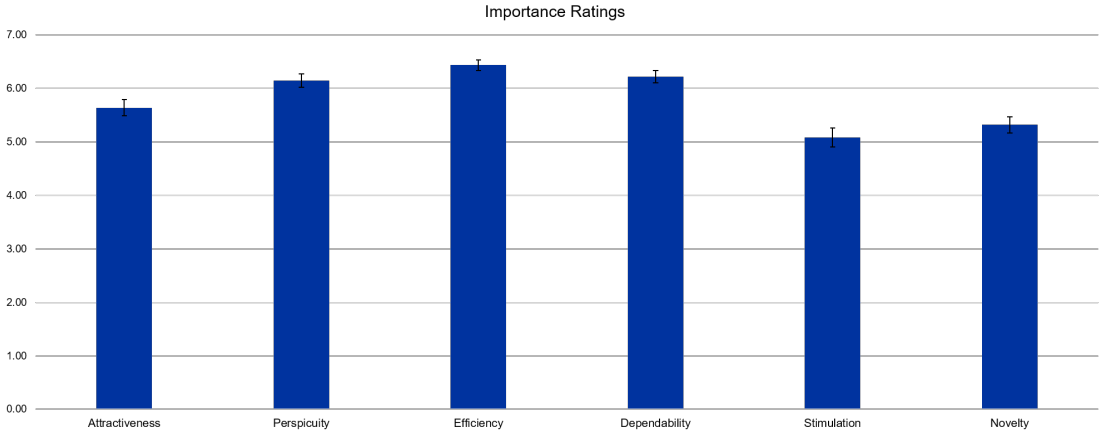


Figure 12: Importance ratings

Also we collected some user feedbacks which were very useful for us to improve the visualisation. From the evaluated feed backs, we conveyed the information that we expected via visualisation.

References

The IUCN Red List of Threatened Species. (n.d.). Retrieved 2021-05-22, from https://www.iucnredlist.org/en

The IUCN Red List of Threatened Species search page. (n.d.). Retrieved 2021-05-22, from https://www.iucnredlist.org/search

What is Information Visualization? (n.d.). Retrieved 2021-05-21, from https://www.interaction-design.org/literature/topics/information-visualization

What is the IUCN Red List? Here's everything you need to know. (n.d.). Retrieved 2021-05-21, from https://www.discoverwildlife.com/animal-facts/what-is-the-iucn-red-list/

10 Annex

10.1 Data tables

Name	Loading	Entries	Exterior	Interior	Points
AMPHIBIANS	15.7 s	8598	142130	87042	62232137
ANGELFISH	2.6 s	86	1421	129093	5970062
BLENNIES	12.1 s	929	23535	546294	29508331
BONEFISH TARPONS	1.6 s	25	19346	36146	2420381
BUTTERFLYFISH	5.6 s	128	3123	267144	12369529
CLUPEIFORMES	7.6 s	468	24800	305511	18140124
GROUPERS	10.3 s	166	10152	339325	25506736
HAGFISH	0.1 s	75	1374	5414	344945
MARINEFISH PART1	50.0 s	1454	141904	1486268	86034377
MARINEFISH PART2	47.8 s	1453	143747	1535923	86210167
MARINEFISH PART3	45.3 s	1454	132248	1486673	82924429
PUFFERFISH	7.6 s	206	29321	237769	14869441
SEABREAMS PORGIES PICARELS	2.3 s	160	1929	87748	7691405
SHARKS RAYS CHIMAERAS	49.5 s	1216	257696	812958	53003848
SURGEONFISHES TANGS UNICORNFISHES	5.1 s	86	3411	236303	10901138
SYNGNATHIFORM FISHES	11.4 s	334	45234	454572	23549776
TUNAS BILLFISHES	7.4 s	62	2889	197336	11955011
WRASSES PARROTFISHES	18.0 s	622	10369	855194	40789728
FW GROUP PART1	67.4 s	7192	1015501	656834	123017192
FW GROUP PART2	55.3 s	7192	959343	631821	112134102
FW GROUP PART3	58.4 s	7192	1011877	622370	125543863
FW GROUP PART4	58.8 s	7192	939868	601965	108204854
FW GROUP PART5	61.1 s	7193	990894	663956	115939669
FW GROUP PART6	63.5 s	7192	959162	684093	108228198
MAMMALS	39.5 s	12894	130575	548268	98208061
MAMMALS FRESHWATER	1.3 s	250	8985	25114	4655065
CONUS	18.8 s	639	8177	863363	42308561
LOBSTERS	10.3 s	251	6758	218283	20779677
MANGROVES	5.0 s	67	16487	201881	9495970
REEF FORMING CORALS PT1	23.7 s	281	9732	1070208	48524496
REEF FORMING CORALS PT2	24.9 s	281	9605	1066327	48712155
REEF FORMING CORALS PT3	24.6 s	281	9908	1035423	47280491
SEAGRASSES	3.1 s	74	1768	107786	6143674
BIRCHES	0.0 s	13	441	5	143003
MAGNOLIAS	0.1 s	129	2170	148	665117
MAPLES	0.1 s	18	1989	7	423162
TEAS	0.0 s	20	101	4	44330
REPTILES	45.0 s	10890	917640	612304	92252623
Total	860.9 s	86763	7995610	18716873	1687125828

Table 4: Loading the whole database (Exterior,Interior = polygons)

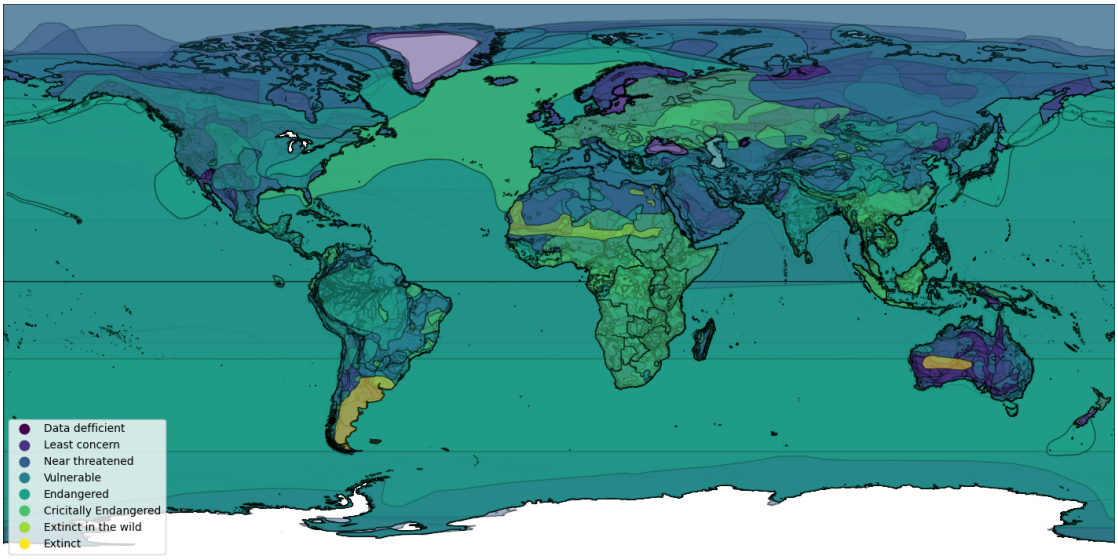


Figure 13: Mammals: Raw data from the REDLIST database (maplotlib default colors viridis)
12895 entries, 5844 species, 1 minute to load, 15 minutes to plot (maplotlib), 1.5GB of data
98 208 061 points, 130 575 exterior polygons, 548 268 interior polygons

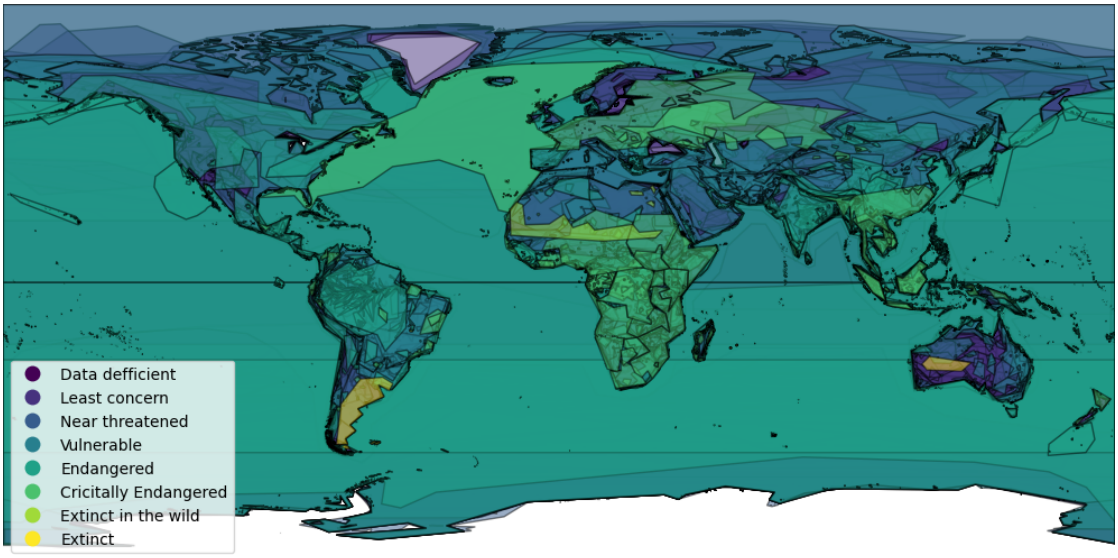


Figure 14: Mammals: All data after simplify(1)
12894 entries, 15 seconds to load, 1 minute to plot (maplotlib), 64.4MB of data
3 486 364 points, 171 174 exterior polygons, 507 669 interior polygons

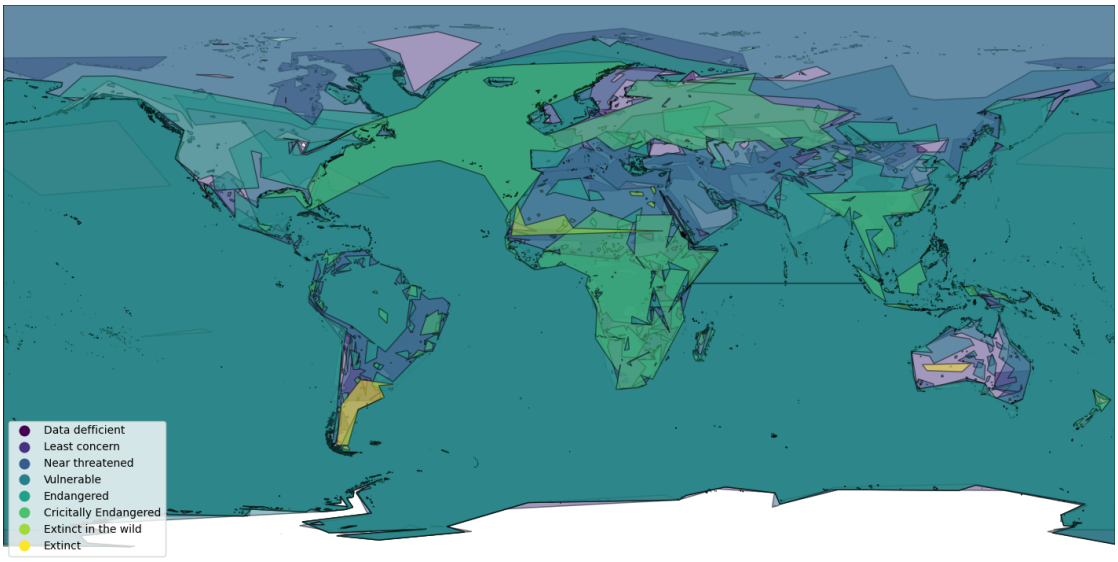


Figure 15: Mammals: Aggregated database after passing trough simplify(3)
8 entries (one per danger level), 1.9 seconds to load, 3 to plot, 6.3MB of data
391440 points, 7 223 exterior polygons, 73 812 interior polygons
Note: it still takes more than 10 seconds to plot in plotly

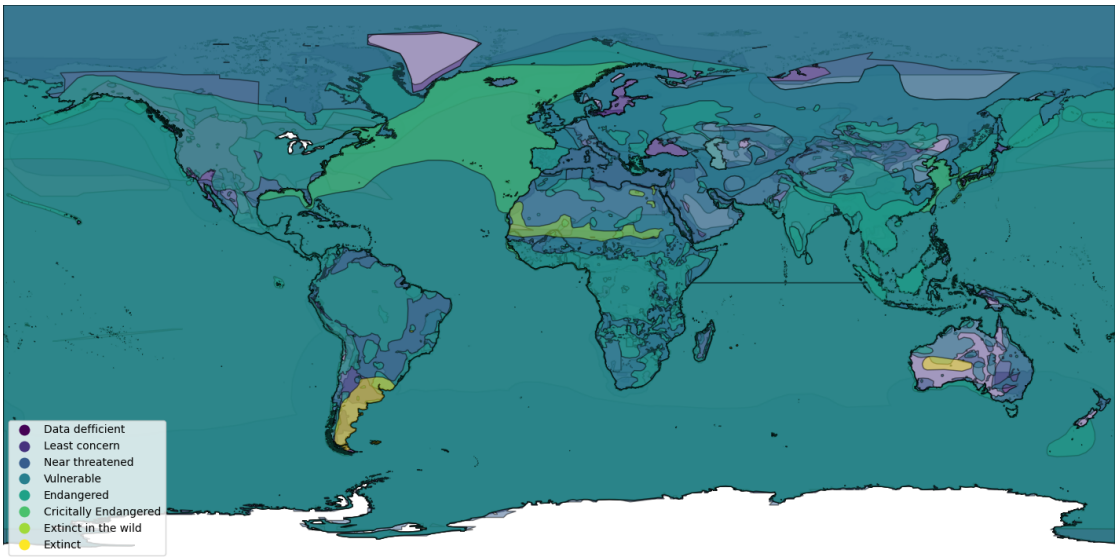


Figure 16: Mammals: Aggregated after deleting small geometry (area<0.05) and simplify(0.1)
8 entries (one per danger level), 1740 shapes, 2.3MB of data

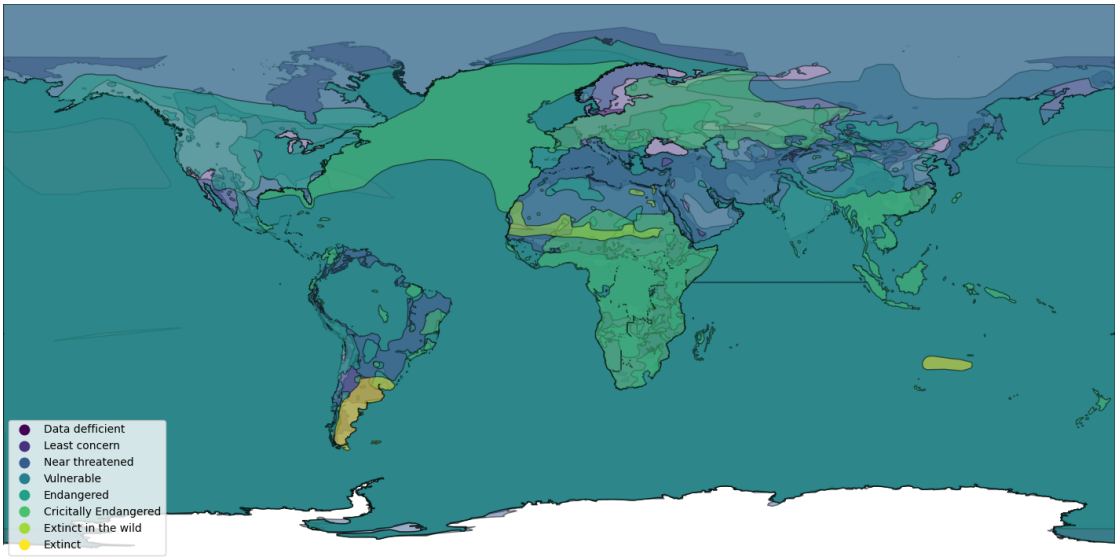


Figure 17: Mammals: Aggregated, remove island<0.05, remove holes, simplify(0.1)
This version is 1.2 MB and is fast-enough to draw into plotly under 10 seconds.
77059 points, 6062 exterior polys, 362 interior polys
Removing holes in addition to small geometries makes the aggregation process a lot faster, but more data is lost (australia). The process should be limited to small holes.

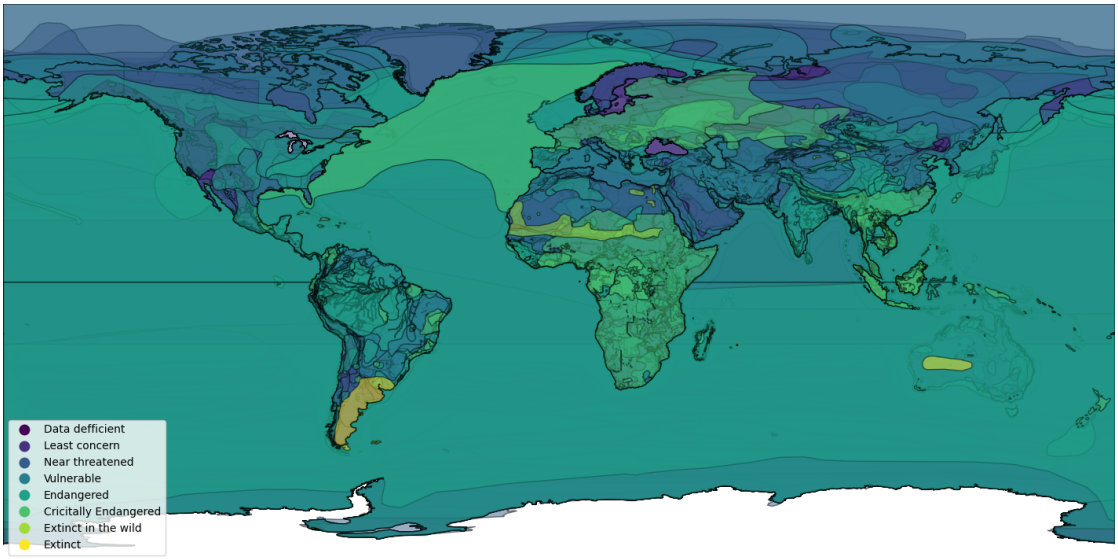


Figure 18: Mammals: Full data, remove islands<0.05, remove holes, simplify(0.1)
This version is 20 MB and plots instantly on Maplotlib. Not on Plotly. 364581 points, 18560 exterior polys, 3999 interior polys 5333 species (some small area species were lost)

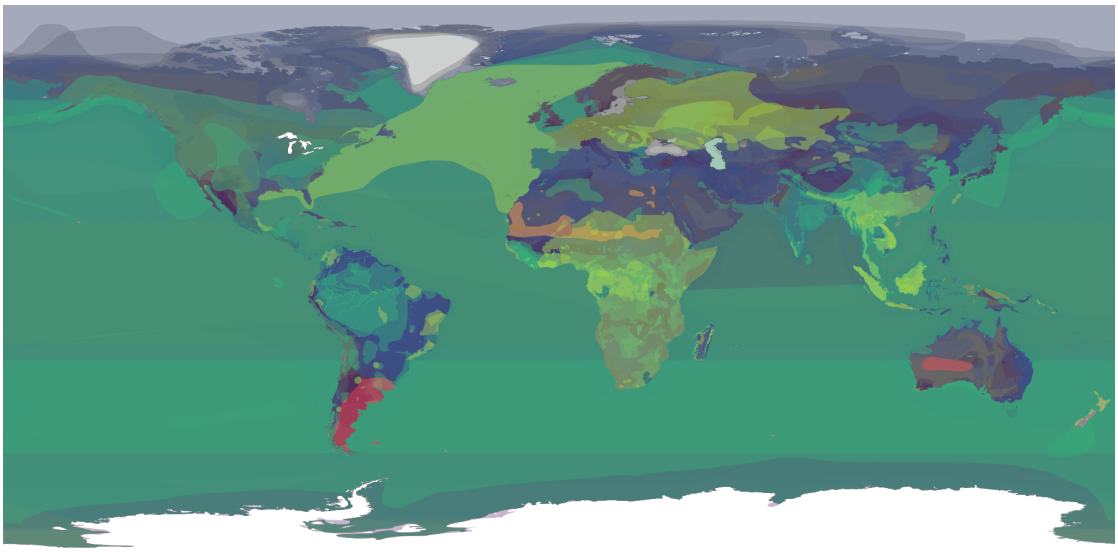


Figure 19: Mammals: Rasterised. Made of 8 transparent PNG layers superposed (custom colors) 8 png at 300DPI (1830x914 px), optimised with PNGcrush, totaling 707 KB

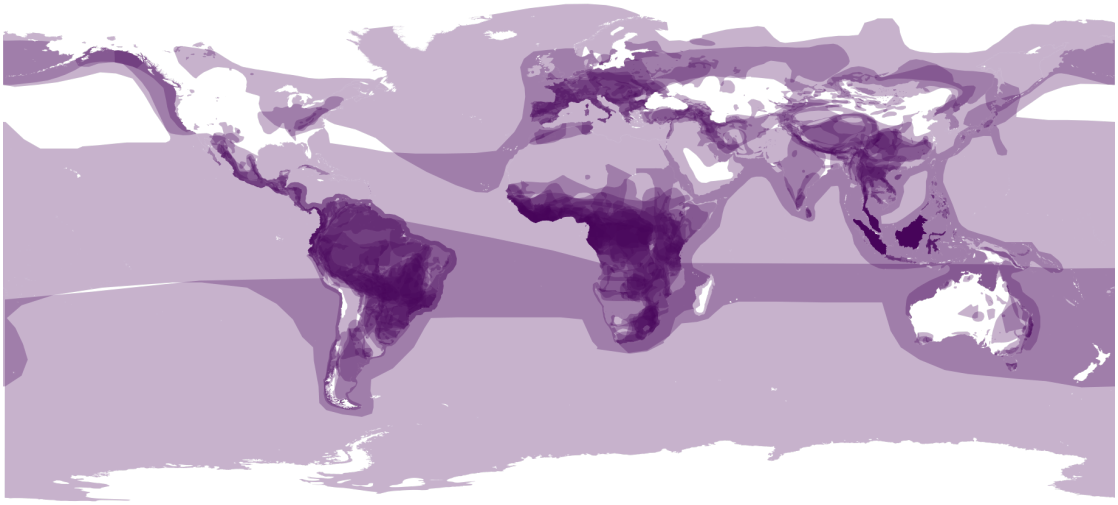


Figure 20: One of the layers used in the rasterised version
One advantage compared to polygonal tables is that we can still see which area have more animals, without increasing the data cost that much (bitwise to 8-bit).